

PyCaret

⌚ Created	@November 25, 2024 10:35 AM
🏷️ Tags	

PyCaret is an open-source, low-code machine learning library in Python that automates machine learning workflows.

[Pycaret?](#)

[Pycaret facettes](#)

[exploratory data analysis](#)

[plot_model](#)

[evaluate_model](#)

[dashboard](#)

[check_fairness](#)

[get_leaderboard](#)

[assign_model](#)

[data preprocessing](#)

[Data Preparation](#)

[Scale and Transform](#)

[Feature Engineering](#)

[Feature Selection](#)

[Other Setup Parameters](#)

[model training](#)

[Compare_models](#)

[Create_model](#)

[model explainability](#)
[interpret_model](#)
[Deployment](#)
[predict_model](#)
[Finalize_model](#)
[Deploy_model](#)
[save_model](#)
[check_drift](#)
[convert_model](#)
[create_api](#)
[create_docker](#)

[License](#)
[cheat sheet](#)

Pycaret?

Pycaret is designed to be an end-to-end machine learning library. This means that is not just made for one specific use case but rather for all your machine learning needs. Under this text is a list of stages that Pycaret has integrated tools for:

- **Data Preparation:** PyCaret can handle data preprocessing tasks such as missing value imputation, scaling, encoding categorical variables, and feature engineering. This is done using the setup function, which prepares the data for modeling.
- **Model Training:** PyCaret provides a simple interface to train multiple machine learning models with just a few lines of code. You can use functions like compare_model to train and evaluate different models quickly.
- **Model Evaluation:** After training, PyCaret offers various tools to evaluate model performance using metrics and visualizations. This helps in understanding how well the models are performing.
- **Model Tuning:** PyCaret includes functionalities for hyperparameter tuning to optimize model performance.
- **Model Deployment:** Once a model is trained and evaluated, PyCaret can help deploy the model into production. It supports exporting models as APIs or saving them for later use.

- **Model Monitoring:** PyCaret integrates with tools like MLflow for tracking experiments and monitoring model performance over time.

Pycaret facettes

exploratory data analysis

Pycaret includes different functions for analysing your dataset(s). In the following section we will explore the 8 functions that they provide for exploring your data easier.

these are functions to be used on trained models

plot_model

This function can be used for analysing the performance of a trained model on the *hold-out set. This function can be used with some parameters to change the resulting graph.

standard example:

```
# load dataset
from pycaret.datasets import get_data
diabetes = get_data('diabetes')

# init setup
from pycaret.classification import *
clf1 = setup(data = diabetes, target = 'Class variable')

# creating a model
lr = create_model('lr')

# plot model
plot_model(lr, plot = 'auc')
```

We first can change the output resolution for the output graphs using the `scale` parameter and later save the output using the 'save = true' parameter in the plot_model() function. Important to know, the library uses Yellowbrick for

plotting, this means that all the arguments that you can pass to Yellowbricks visualiser are acceptable for Pycaret aswell. Lastly, you can use train data aswell by passing 'use_train_data=true' in the plot_model() function.

Keep in mind, using train data where the model was already trained on may conclude to a biased result.

when passing plot='xxx' we should give a module for what we want to analyse. For a list of all the different modules, please reference the pycaret documentation with the following link.

link: <https://pycaret.gitbook.io/docs/get-started/functions/analyze#examples-by-module>

*A hold-out set is used to give a objective review of the model's performance during the training process.

evaluate_model

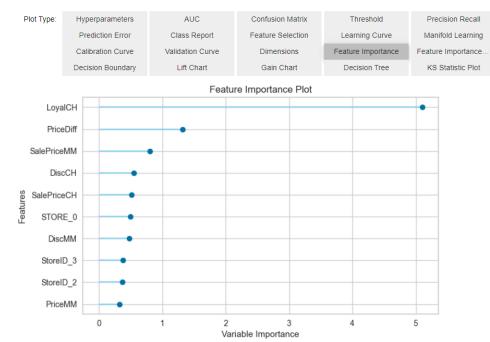
this function is used for analysing the performance of the model as well but displays a user interface instead. It calls the plot_model user interface internally and can only be run in a jupyter notebook or similar environment.

```
# load dataset
from pycaret.datasets import get
juice = get_data('juice')

# init setup
from pycaret.classification import
exp_name = setup(data = juice,

# create model
lr = create_model('lr')

# launch evaluate widget
evaluate_model(lr)
```



dashboard

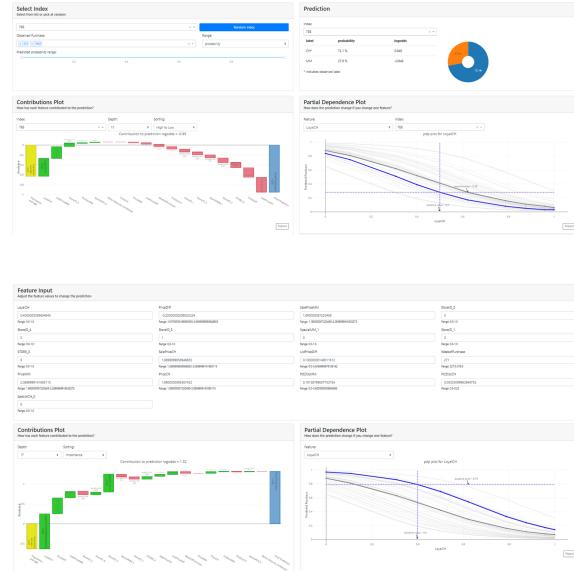
'Dashboard' function is used for generating interactive user interface dashboard. You can input certain values and the plots will change accordingly. This function uses ExplainerDashboard, this is another library used for building interactive dashboard.

```
# load dataset
from pycaret.datasets import
juice = get_data('juice')

# init setup
from pycaret.classification
exp_name = setup(data = juice)

# train model
lr = create_model('lr')

# launch dashboard
dashboard(lr)
```



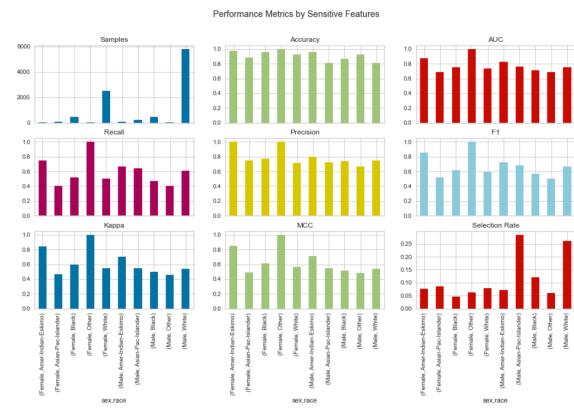
check_fairness

This function is used to check the fairness of a model. It evaluates the model's predictions or decisions and sees if it impacts different groups of individuals. It helps insure that the model does not inherently have a bias towards a particular group. 'check_fairness' function gives back fairness-related metrics between different groups.

```
# load dataset
from pycaret.datasets import
income = get_data('income')

# init setup
from pycaret.classification
exp_name = setup(data = income)

# train model
```



```

lr = create_model('lr')

# check model fairness
lr_fairness = check_fairness

```

		Samples	Accuracy	AUC	Recall	Precision	F1	Kappa	MCC	Selection Rate
sex	race									
Female	Amer-Indian-Eskimo	39	0.974359	0.875	0.75	1.0	0.857143	0.843373	0.853913	0.076923
	Asian-Pac-Islander	94	0.882979	0.687342	0.4	0.75	0.521739	0.462019	0.491733	0.085106
	Black	473	0.955603	0.751894	0.515152	0.772727	0.618162	0.595611	0.609441	0.046512
	Other	32	1.0	1.0	1.0	1.0	1.0	1.0	1.0	0.0625
Male	White	2523	0.923107	0.74021	0.655376	0.715736	0.592437	0.551373	0.561562	0.078082
	Amer-Indian-Eskimo	70	0.957143	0.82521	0.666667	0.6	0.727273	0.704225	0.707673	0.071429
	Asian-Pac-Islander	218	0.807339	0.763996	0.642857	0.725806	0.681818	0.544387	0.546449	0.284404
	Black	470	0.865957	0.713596	0.466667	0.736842	0.571429	0.496685	0.514902	0.121277
Male	Other	51	0.921569	0.68913	0.4	0.666667	0.5	0.460317	0.478051	0.058024
	White	5799	0.808243	0.755036	0.606337	0.748674	0.67003	0.536974	0.542904	0.260045

get_leaderboard

This function simply returns a table with all the models that you are trying to evaluate and rank them by performance.

```

# load dataset
from pycaret.datasets import
diabetes = get_data('diabete

# init setup
from pycaret.classification
clf1 = setup(data = diabetes

# compare models
top3 = compare_models(n_sele

# tune top 3 models
tuned_top3 = [tune_model(i)

# ensemble top 3 tuned model
ensembled_top3 = [ensemble_m

# blender
blender = blend_models(tuned

# stacker
stacker = stack_models(tuned

# check leaderboard
get_leaderboard()

```

Index	Model Name	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	Logistic Regression	(DataTypes_Auto_infer(categorical_features=[]))	0.7598	0.8143	0.5229	0.6884	0.5875	0.4230	0.4355
1	K Neighbors Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7321	0.7723	0.5291	0.6136	0.5629	0.3723	0.3780
2	Naive Bayes	(DataTypes_Auto_infer(categorical_features=[]))	0.6725	0.7161	0.1542	0.5483	0.2381	0.0991	0.1377
3	Decision Tree Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.6688	0.6324	0.5275	0.5862	0.5077	0.2806	0.2675
4	SVM - Linear Kernel	(DataTypes_Auto_infer(categorical_features=[]))	0.5715	0.0000	0.5729	0.3904	0.4177	0.1314	0.1526
5	Ridge Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7561	0.0000	0.4888	0.7045	0.5678	0.4056	0.4247
6	Random Forest Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7596	0.8116	0.5062	0.6829	0.5700	0.3993	0.4089
7	Quadratic Discriminant Analysis	(DataTypes_Auto_infer(categorical_features=[]))	0.5195	0.5559	0.5788	0.7377	0.4039	0.0636	0.0733
8	Ada Boost Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7449	0.7787	0.5735	0.6253	0.5980	0.4104	0.4127
9	Gradient Boosting Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7300	0.8048	0.5595	0.5940	0.5944	0.3744	0.3784
10	Linear Discriminant Analysis	(DataTypes_Auto_infer(categorical_features=[]))	0.7487	0.8099	0.4830	0.6836	0.5580	0.3896	0.4064
11	Extra Trees Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7432	0.7855	0.4892	0.6545	0.5588	0.3808	0.3913
12	Extreme Gradient Boosting	(DataTypes_Auto_infer(categorical_features=[]))	0.7301	0.7752	0.5631	0.5937	0.5734	0.3776	0.3807
13	Light Gradient Boosting Machine	(DataTypes_Auto_infer(categorical_features=[]))	0.7319	0.7844	0.5565	0.6055	0.5747	0.3802	0.3846
14	CatBoost Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7506	0.8224	0.5516	0.6424	0.5897	0.4128	0.4179
15	Dummy Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.6723	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000
16	Logistic Regression	(DataTypes_Auto_infer(categorical_features=[]))	0.7691	0.8178	0.5348	0.7108	0.6030	0.4451	0.4590
17	Ridge Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7580	0.0000	0.4841	0.7056	0.5720	0.4112	0.4294
18	Random Forest Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7727	0.8152	0.8056	0.6206	0.6989	0.5216	0.5357
19	Logistic Regression	(DataTypes_Auto_infer(categorical_features=[]))	0.7543	0.8194	0.5288	0.6829	0.5837	0.4133	0.4215
20	Ridge Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7653	0.7441	0.4888	0.7156	0.5725	0.4205	0.4394
21	Random Forest Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7653	0.8179	0.7719	0.6145	0.6819	0.4998	0.5110
22	Voting Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7636	0.0000	0.5288	0.7043	0.5955	0.4333	0.4481
23	Stacking Classifier	(DataTypes_Auto_infer(categorical_features=[]))	0.7580	0.8232	0.5180	0.6767	0.5828	0.4173	0.4271

assign_model

This function gives back a table and assigns labels to the training dataset using the trained model. This is available for following *modules:

- clustering
- anomaly_detection
- Natural Language Processing

```
# load dataset
from pycaret.datasets import
jewellery = get_data('jewellery')

# init setup
from pycaret.clustering import
# or this when using anomaly
# from pycaret.clustering import

clu1 = setup(data = jewellery)

# train a model
kmeans = create_model('kmean')

# assign model
assign_model(kmeans)
```

	Age	Income	SpendingScore	Savings	Cluster
0	58	77769	0.791329	6559.829923	Cluster 2
1	59	81799	0.791082	5417.661426	Cluster 2
2	62	74751	0.702657	9258.992965	Cluster 2
3	59	74373	0.765680	7346.334504	Cluster 2
4	87	17760	0.348778	16869.507130	Cluster 1
...
500	28	101206	0.387441	14936.775389	Cluster 3
501	93	19934	0.203140	17969.693769	Cluster 1
502	90	35297	0.355149	16091.401954	Cluster 1
503	91	20681	0.354679	18401.088445	Cluster 1
504	89	30267	0.289310	14386.351880	Cluster 1

505 rows × 5 columns

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Anomaly	Anomaly_Score
0	0.263995	0.764929	0.138424	0.935242	0.005987	0.516790	0.912225	0.608234	0.723782	0.733991	0	-0.023131
1	0.546092	0.653975	0.065575	0.227772	0.845269	0.837066	0.272379	0.331679	0.429297	0.367422	0	-0.077484
2	0.336714	0.538342	0.192301	0.553563	0.074515	0.332093	0.365792	0.861309	0.899017	0.088600	0	-0.002603
3	0.092108	0.995017	0.014465	0.176371	0.241530	0.514724	0.562208	0.158963	0.073715	0.208463	1	0.060657
4	0.325261	0.805988	0.957033	0.331665	0.307923	0.355315	0.501899	0.558449	0.888169	0.182754	0	-0.011531
...
995	0.305055	0.656837	0.331665	0.822525	0.907127	0.882278	0.855732	0.584786	0.809840	0.242762	0	-0.076773
996	0.012027	0.884258	0.616604	0.167966	0.811223	0.930071	0.418462	0.472306	0.348347	0.671129	0	-0.084505
997	0.250967	0.138927	0.919703	0.461234	0.888555	0.869888	0.800908	0.530324	0.779433	0.234952	0	-0.054988
998	0.502436	0.938820	0.580062	0.540773	0.151995	0.059452	0.225220	0.242755	0.279388	0.538755	0	-0.087168
999	0.457091	0.017755	0.714113	0.125992	0.063316	0.154739	0.92974	0.692299	0.816777	0.307592	0	-0.016641

1000 rows × 12 columns

*modules: these are machine learning use-cases that are supported in PyCaret

data preprocessing

Data Preparation

- **Missing values:** fill in empty records encoded as blanks or NaN (not a number).
- **Data Types:** it automatically detects the data types of each feature

- **One-Hot Encoding:** it gives labels like ordinal or nominal instead of continuous numbers
- **Ordinal Encoding:** it encodes the categorical features with a natural hierarchy differently
ex. labels like: Low, Medium, High,...
- **Cardinal Encoding:** this encoding should be used with high cardinality categorical features in datasets
- **Target Imbalance:** when the training dataset has unequal distribution of target class
- **Remove Outliers:** you can remove outliers from the training dataset

Scale and Transform

- **Normalize**
- **Feature Transform:** change the shape of the distribution of features
- **Target Transform:** change the shape of the distribution of the target variables

Feature Engineering

- **Feature Interaction:** Combining two features through mathematical operations can often explain variance better than the features individually.
- **Polynomial Features:** Introduces non-linearity by creating higher-order terms of existing features.
- **Group Features:** Calculate group statistics (mean, median, variance, standard deviation) for related features.
- **Feature Binning:** Converts continuous variables into categorical ones by dividing them into predefined intervals (bins).
- **Combine Rare Levels:** Merges rare levels in categorical features with high cardinality to reduce sparsity.
- **Create Clusters:** Uses unsupervised ML techniques to create new features based on clusters in the data.

Feature Selection

- **Feature Selection:** Identifies the most relevant features for predicting the target variable, reducing overfitting, improving accuracy, and speeding up training.
- **Remove Multicollinearity:** Eliminates features that are highly linearly correlated with each other to avoid redundancy and instability.
- **Principal Component Analysis (PCA):** An unsupervised technique to reduce dimensionality by compressing the feature space while preserving variance.
- **Ignore Low Variance:** Discards features with skewed distributions or levels dominated by one category, as they provide minimal predictive power.

Other Setup Parameters

- **Required Parameters:**
 - **Data:** The dataset to be used.
 - **Target Variable:** The name of the target column.
- **Experiment Logging:** Tracks metrics, hyperparameters, and artifacts automatically using **MLflow**.
- **Model Selection:** Allows configuration of parameters that influence the model selection process, separate from data preprocessing.
- **Miscellaneous Parameters:** Control settings like GPU usage for training and verbosity levels of the experiment.

model training

Compare_models

This function trains and evaluates the performance of all estimators available in the model library using cross-validation. The output of this function is a scoring grid with average cross-validated scores.

```
# load dataset
from pycaret.datasets import
diabetes = get_data('diabete

# init setup
from pycaret.classification
clf1 = setup(data = diabetes

# compare models
best = compare_models()
```

Model		Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC	TT (Sec)
catboost	CatBoost Classifier	0.7672	0.8324	0.5681	0.6901	0.6177	0.4537	0.4616	0.6180
rf	Random Forest Classifier	0.7635	0.8121	0.5184	0.7051	0.5904	0.4315	0.4452	0.0720
et	Extra Trees Classifier	0.7597	0.7945	0.5132	0.7024	0.5851	0.4236	0.4378	0.0650
lr	Logistic Regression	0.7526	0.8175	0.5243	0.6701	0.5806	0.4115	0.4211	0.7690
knn	K Neighbors Classifier	0.7525	0.7543	0.5740	0.6572	0.6077	0.4290	0.4348	0.0090
xgboost	Extreme Gradient Boosting	0.7502	0.8080	0.6178	0.6386	0.6236	0.4374	0.4413	0.1030
ridge	Ridge Classifier	0.7490	0.0000	0.5132	0.6660	0.5719	0.4011	0.4114	0.0030
lda	Linear Discriminant Analysis	0.7490	0.8112	0.5298	0.6586	0.5791	0.4059	0.4144	0.0040
lightgbm	Light Gradient Boosting Machine	0.7448	0.8053	0.6009	0.6324	0.6109	0.4220	0.4268	0.0530
gbc	Gradient Boosting Classifier	0.7411	0.8186	0.5515	0.6403	0.5844	0.3995	0.4067	0.0310
ada	Ada Boost Classifier	0.7392	0.7894	0.5406	0.6323	0.5789	0.3928	0.3976	0.0270
dt	Decision Tree Classifier	0.7058	0.6705	0.5629	0.5609	0.5583	0.3388	0.3411	0.0040
dummy	Dummy Classifier	0.6629	0.5000	0.0000	0.0000	0.0000	0.0000	0.0000	0.0030
nb	Naive Bayes	0.6610	0.7321	0.2260	0.5176	0.3045	0.1245	0.1477	0.0040
qda	Quadratic Discriminant Analysis	0.5201	0.5868	0.6035	0.3977	0.4553	0.0870	0.0692	0.0050
svm	SVM - Linear Kernel	0.5197	0.0000	0.5000	0.2441	0.3097	0.0211	0.0194	0.0050

Create_model

This function trains and evaluates the performance of a given estimator using cross-validation. The output of this function is a scoring grid with CV scores by fold.

```
# load dataset
from pycaret.datasets import
diabetes = get_data('diabete

# init setup
from pycaret.classification
clf1 = setup(data = diabetes

# train logistic regression
lr = create_model('lr')
```

	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC
0	0.8148	0.8827	0.5263	0.9091	0.6667	0.5507	0.5902
1	0.8148	0.8451	0.5263	0.9091	0.6667	0.5507	0.5902
2	0.7778	0.8556	0.6842	0.6842	0.6842	0.5128	0.5128
3	0.6296	0.6737	0.3684	0.4667	0.4118	0.1469	0.1491
4	0.7037	0.7865	0.5263	0.5882	0.5556	0.3344	0.3355
5	0.8148	0.8932	0.6316	0.8000	0.7059	0.5735	0.5820
6	0.7778	0.8406	0.6316	0.7059	0.6667	0.5008	0.5025
7	0.8113	0.8190	0.5556	0.8333	0.6667	0.5423	0.5640
8	0.7736	0.8607	0.6842	0.6842	0.6842	0.5077	0.5077
9	0.7925	0.8901	0.5789	0.7857	0.6667	0.5210	0.5338
Mean	0.7711	0.8347	0.5713	0.7366	0.6375	0.4741	0.4868
SD	0.0570	0.0621	0.0896	0.1329	0.0842	0.1259	0.1330

model explainability

interpret_model

This function is used for analysing the predictions generated by the model. Most of the graphs are based on 'Shapley Additive exPlanations' or SHAP. It uses a game theory approach to explain the output of machine learning models.

```

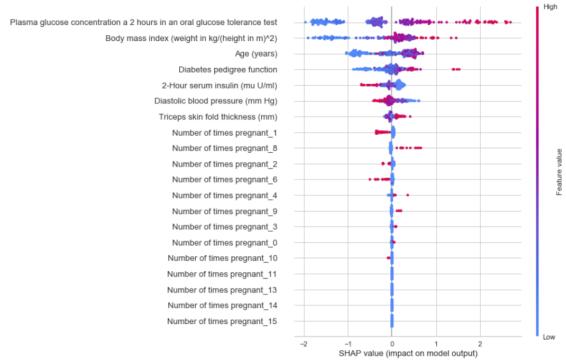
# load dataset
from pycaret.datasets import
diabetes = get_data('diabete

# init setup
from pycaret.classification
clf1 = setup(data = diabetes

# creating a model
xgboost = create_model('xgb

# interpret model
interpret_model(xgboost)

```



Just like with the function 'model_plot' you can use 'save=true' as a parameter to save the output as png and change te plot type with 'plot=xxx'. The following plots can be displayed:

- **correlation**: Helps in understanding how features are related to each other and to the target variable. Strong correlations can indicate multicollinearity, which might affect the model's performance.
- **partial dependency plot**: Useful for understanding the relationship between a feature and the target variable, while averaging out the effects of all other features.
- **Morris sensitivity Analysis**: Helps in identifying which features have the most significant impact on the model's output, which is crucial for model interpretation and feature selection.
- **Permutation Feature Importance**: Useful for understanding the contribution of each feature to the model's predictions. It is model-agnostic and can be applied to any machine learning model.
- **Reason Plot**: Helps in explaining why the model made a specific prediction for a particular instance, which is important for model transparency and trust.

Deployment

predict_model

this function generates labels using a trained model

```
# load dataset
from pycaret.datasets import get_data
diabetes = get_data('diabetes')

# init setup
from pycaret.classification import setup
clf1 = setup(data = diabetes)

# create a model
xgboost = create_model('xgboost')

# predict on hold-out
predict_model(xgboost)
```

	Model	Accuracy	AUC	Recall	Prec.	F1	Kappa	MCC					
0	Extreme Gradient Boosting	0.7143	0.7848	0.5301	0.6197	0.5714	0.3591	0.3615					
	Age	Number of times pregnant	Number of times pregnant,1	Number of times pregnant,10	Number of times pregnant,2	Number of times pregnant,3	Number of times pregnant,4	Number of times pregnant,5	Number of times pregnant,6	Number of times pregnant,7	Number of times pregnant,8	Number of times pregnant,9	
	(years)												
23.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0	1 0.9994
55.0	0.0	0.0	0.0	...	0.0	0.0	0.0	1.0	0.0	0.0	0.0	0	1 0.9136
46.0	0.0	1.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	1 0.9831
25.0	0.0	0.0	0.0	...	1.0	0.0	0.0	0.0	0.0	0.0	0.0	1	1 0.9983
21.0	1.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	1	1 0.8417
...	
33.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	1	1 0.8881
24.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0 0.9994
51.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0	0 0.9848
28.0	0.0	0.0	0.0	...	0.0	1.0	0.0	0.0	0.0	0.0	0.0	0	0 0.9999
47.0	0.0	0.0	0.0	...	0.0	0.0	0.0	0.0	1.0	0.0	0.0	1	1 0.7852

Finalize_model

This function trains a given model on the entire dataset including the hold-out set.

```
# load dataset
from pycaret.datasets import get_data
diabetes = get_data('diabetes')

# init setup
from pycaret.classification import *
clf1 = setup(data = diabetes, target = 'Class variable')

# create a model
rf = create_model('rf')

# finalize a model
finalize_model(rf)
```

Deploy_model

deploys entire ML pipeline on the cloud

```
# load dataset
from pycaret.datasets import get_data
diabetes = get_data('diabetes')

# init setup
from pycaret.classification import *
clf1 = setup(data = diabetes, target = 'Class variable')

# create a model
lr = create_model('lr')

# finalize a model
final_lr = finalize_model(lr)

# deploy a model
deploy_model(final_lr, model_name = 'lr_aws', platform = 'aws')
```

Services

- **AWS (amazon web services) environment variables**
 - AWS Access Key ID
 - AWS Secret Key Access
 - Default Region Name (can be seen under Global settings on your AWS console)
 - Default output format (must be left blank)
- **GCP (google cloud platform)**
 - the project must be created using the command-line or GCP console. Once the project is created, you must create a service account and download the service account key as a JSON file to set environment variables in your local environment.
- **Azure**
 - environment variables for the connection string must be set in your local environment. Go to settings of storage account on Azure portal to access the connection string required.

save_model

This function saves the transformation pipeline and a trained model object into the current working directory as a pickle file for later use.

```
# load dataset
from pycaret.datasets import get_data
diabetes = get_data('diabetes')

# init setup
from pycaret.classification import *
clf1 = setup(data = diabetes, target = 'Class variable')

# create a model
dt = create_model('dt')

# save pipeline
save_model(dt, 'dt_pipeline')
```

check_drift

generates a drift report file. Model drift refers to the degradation of machine learning model performance due to changes in data or in the relationships between input and output variables. Model drift—also known as model decay—can negatively impact model performance, resulting in faulty decision-making and bad predictions.

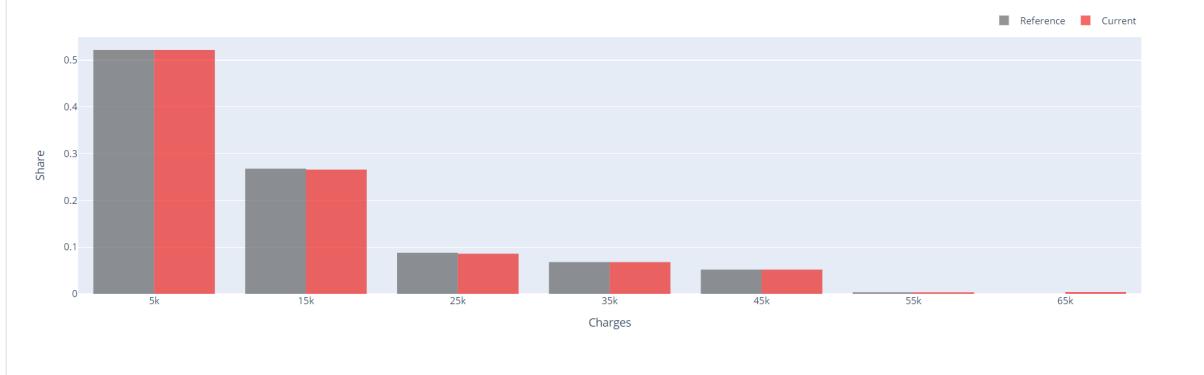
```
# load dataset
from pycaret.datasets import get_data
data = get_data('insurance')

# generate drift report
check_drift(reference_data = data.head(500), current_data = d
```

Drift is detected for 0.00% of features (0 out of 7). Dataset Drift is NOT detected.

Feature	Type	Reference Distribution	Current Distribution	Data Drift	Stat Test	Drift Score
> charges	num			Not Detected	K-S p_value	0.770437
> children	num			Not Detected	K-S p_value	0.935112
> sex	cat			Not Detected	Z-test p_value	0.89933
> smoker	cat			Not Detected	Z-test p_value	0.64363
> region	cat			Not Detected	chi-square p_value	0.630943
> age	num			Not Detected	K-S p_value	0.329358
> bmi	num			Not Detected	K-S p_value	0.172556

Target Drift: detected, drift score=0.0 (chi-square p_value)



convert_model

This function transpiles the trained machine learning model's decision function in different programming languages. Useful if you want to deploy models into environments where you can't install your normal Python stack to support model inference.

```
# load dataset
from pycaret.datasets import get_data
juice = get_data('juice')

# init setup
from pycaret.classification import *
exp_name = setup(data = juice, target = 'Purchase')

# train a model
lr = create_model('lr')
```

```
# convert a model  
convert_model(lr, 'java')
```

create_api

This function takes an input model and creates a POST API for inference. It only creates the API and doesn't run it automatically. To run the API, you must run the Python file using

```
# load dataset  
from pycaret.datasets import get_data  
juice = get_data('juice')  
  
# init setup  
from pycaret.classification import *  
exp_name = setup(data = juice, target = 'Purchase')  
  
# train a model  
lr = create_model('lr')  
  
# create api  
create_api(lr, 'lr_api')  
  
# run api  
!python lr_api.py
```

create_docker

This function creates a Dockerfile and requirements.txt for productionalizing API end-point.

```
# load dataset  
from pycaret.datasets import get_data  
juice = get_data('juice')  
  
# init setup  
from pycaret.classification import *  
exp_name = setup(data = juice, target = 'Purchase')
```

```

# train a model
lr = create_model('lr')

# create api
create_api(lr, 'lr_api')

# create docker
create_docker('lr_api')

```

License

Pycaret is distributed under the MIT license which means you can use pycaret free of charge and are allowed to sell.

cheat sheet

	Regression & Classification	Time Series	Clustering	Anomaly Detection
Tutorials Clustering Beginner Anomaly Beginner Association Rules Beginner NLP Beginner Intermediate Regression Beginner Intermediate Classification Binary (Beginner) Binary (Intermediate) Multiclass (Beginner) Time Series Beginner Supervised Learning Times Series Analysis Unsupervised Learning <small>(1) classification only</small>	<pre> setup() create_model(*) compare_models() ensemble_models() tune_model() blend_models() stack_models() plot_model(**) evaluate_model() interpret_model() calibrate_model() optimize_threshold() predict_model() finalize_model() deploy_model() deep_check() save_model() load_model() automl() pull() models() get_metrics() add_metric() remove_metric() get_logs() get_config() set_config() save_experiment() load_experiment() get_leaderboard() get_current_experiment() get_current_experiment() convert_model() eda() check_fairness() create_api() create_docker() create_app() get_allowed_engines() get_engine() check_drift() </pre>	<pre> * model: (classification) (regression) 'lr' 'lasso' 'knn' 'ridge' 'dt' 'en' 'svm' 'lar' 'rfsvm' 'llar' 'gpc' 'omp' 'mlp' 'pr' 'ridge' 'ard' 'rfe' 'par' 'ada' 'transac' 'ada' 'tr' 'gbc' 'huber' 'dt' 'knn' 'lda' 'et' 'catboost' 'et' 'catboost' 'ada' 'gbn' 'gbm' *x plot= 'threshold' 'xgbosst' 'pr' 'lightgbm' 'error' 'catboost' 'class_report' ** plot= 'boundary' 'residuals' 'rfe' 'error' 'learning' 'cooks' 'manifold' 'rfe' 'calibration' 'learning' 'vc' 'vc' 'dimension' 'manifold' 'feature' 'feature' 'feature_all' 'feature_all' 'parameter' 'residuals_' 'lift' 'interactive' 'gain' 'parameter' 'tree' 'tree' 'ks' 'confusion_matrix' </pre>	<pre> setup() create_model(*) compare_models() tune_model() blend_models() plot_model() predict_model() deploy_model() finalize_model() forecast() insample() load_model() train_test_split() decomp_stl() diagnostics() load() get_forecast() get_residuals() get_train_test_split() decomp_classical() get_models() get_metrics() add_metric() remove_metric() get_logs() get_config() set_config() save_experiment() load_experiment() set_current_experiment() get_current_experiment() get_check_stats() </pre>	<pre> setup() create_model() compare_models() tune_model() blend_models() plot_model() predict_model() deploy_model() finalize_model() load() get_forecast() get_residuals() get_train_test_split() decomp_classical() get_models() get_metrics() add_metric() remove_metric() get_logs() get_config() set_config() save_experiment() load_experiment() set_current_experiment() get_allowed_engines() load_experiment() get_engine() get_current_experiment() get_check_stats() get_allowed_engines() get_engine() </pre>

Parameters of setup() and its *default values*

Regression & Classification

```
pycaret.classification.ClassificationExperiment()
setup(
    data = None,
    data_func = None,
    target = -1,
    index = True,
    train_size = 0.7,
    test_size = None,
    ordinal_features = None,
    numeric_features = None,
    categorical_features = None,
    ignore_features = None,
    text_features = None,
    ignore_features = None,
    keep_features = None,
    preprocess = True,
    create_date_columns = False,
    create_date_columns = ['day','month','year'],
    imputation_type = 'simple',
    numeric_imputation = 'mean',
    categorical_imputation = 'mode',
    iterative_imputation_iters = 5,
    numeric_iterative_imputer = 'lightgbm',
    categorical_iterative_imputer = 'lightgbm',
    text_imputer = 'tf-idf',
    max_encoding_ohc = 25,
    encoding_method = None,
    rare_to_value = 0.05,
    rare_to_value = None,
    group_by_rare = None,
    polynomial_features = False,
    polynomial_degree = 2,
    bin_numeric_features = None,
    remove_outliers = False,
    outlier_method = 'iforest',
    outlier_threshold = 3.5,
    fix_imbalance = False,
    fix_imbalance_method = 'SMOTE',
    transformation = False,
    transform_target_method = 'yeo-johnson',
    normalize = False,
    normalize_method = 'sscore',
    profile_kwarg = None)
```

(c) Classification only

(r) Regression only

Time Series

```
pycaret.time_series.TSForecastingExperiment()
setup(
    data = None,
    data_func = None,
    target = None,
    index = None,
    ignore_features = None,
    numeric_imputation_exogenous = None,
    date_features = None,
    text_features = None,
    ignore_features = None,
    keep_features = None,
    preprocess = True,
    create_date_columns = ['day','month','year'],
    imputation_type = 'simple',
    numeric_imputation = 'mean',
    categorical_imputation = 'mode',
    text_features_method = 'tf-idf',
    fh = 3,
    fh = 1,
    n_jobs = -1,
    session_id = None,
    ignore_seasonality_test = False,
    sp_detection = 'auto',
    max_sp_to_consider = 60,
    remove_outliers = False,
    harmonic_order_method = 'harmonic_max',
    num_sp_to_use = 1,
    polya_alpha = None,
    coverage = 0.95,
    enforce_exogenous = True,
    n_jobs = -1,
    use_gpu = False,
    custom_pipeline = None,
    html = True,
    session_id = None,
    system_log = True,
    log_experiment = False,
    experiment_name = None,
    experiment_custom_tags = None,
    log_plots = False,
    log_data = False,
    log_profile = False,
    log_outliers = False,
    log_outliers_threshold = 0.05,
    engine = None,
    verbose = True,
    memory = True,
    profile = False,
    profile_kwarg = None)
```

(c) Classification only

(r) Regression only

Clustering

```
pycaret.clustering.ClusteringExperiment()
setup(
    data = None,
    data_func = None,
    ordinal_features = None,
    numeric_features = None,
    categorical_features = None,
    date_features = None,
    text_features = None,
    ignore_features = None,
    keep_features = None,
    preprocess = True,
    create_date_columns = ['day','month','year'],
    imputation_type = 'simple',
    numeric_imputation = 'mean',
    categorical_imputation = 'mode',
    text_features_method = 'tf-idf',
    max_encoding_ohc = -1,
    encoding_ohc = None,
    rare_to_value = None,
    rare_value = 'rare',
    polynomial_features = False,
    polynomial_degree = 2,
    low_variance_threshold = None,
    remove_multicollinearity = False,
    multicollinearity_threshold = 0.9,
    bin_numeric_features = None,
    remove_outliers = False,
    outliers_method = 'iforest',
    outliers_threshold = 0.05,
    transformation = False,
    normalization_method = 'yeo-johnson',
    normalize = False,
    normalize_method = 'zscore',
    pca = True,
    pca_method = 'linear',
    pca_components = None,
    custom_pipeline = None,
    custom_pipeline_position = -1,
    n_jobs = -1,
    use_gpu = False,
    html = True,
    session_id = None,
    system_log = True,
    log_experiment = False,
    experiment_name = None,
    experiment_custom_tags = None,
    log_plots = False,
    log_profile = False,
    log_outliers = False,
    log_outliers_threshold = 0.05,
    engine = None,
    verbose = True,
    memory = True,
    profile = False,
    profile_kwarg = None)
```

Anomaly Detection

```
pycaret.anomaly.AnomalyExperiment()
setup(
    data = None,
    data_func = None,
    index = True,
    ordinal_features = None,
    numeric_features = None,
    categorical_features = None,
    date_features = None,
    text_features = None,
    ignore_features = None,
    keep_features = None,
    preprocess = True,
    create_date_columns = ['day','month','year'],
    imputation_type = 'simple',
    numeric_imputation = 'mean',
    categorical_imputation = 'mode',
    text_features_method = 'tf-idf',
    max_encoding_ohc = -1,
    encoding_ohc = None,
    rare_to_value = None,
    rare_value = 'rare',
    polynomial_features = False,
    polynomial_degree = 2,
    low_variance_threshold = None,
    group_name = None,
    drop_low_variance = True,
    remove_multicollinearity = False,
    multicollinearity_threshold = 0.9,
    pca = False,
    pca_method = 'linear',
    pca_components = None,
    custom_pipeline = None,
    custom_pipeline_position = -1,
    n_jobs = -1,
    use_gpu = False,
    html = True,
    session_id = None,
    system_log = True,
    log_experiment = False,
    experiment_name = None,
    experiment_custom_tags = None,
    log_plots = False,
    log_profile = False,
    log_outliers = False,
    log_outliers_threshold = 0.05,
    engine = None,
    verbose = True,
    memory = True,
    profile = False,
    profile_kwarg = None)
```